Introduction to coding

Objectives

1.) Introduction

- orientation in RStudio
- execution command, basic operators
- assigning operator
- vectors
- comments

2.) Syntax and basic functions

- functions, objects, values, syntax
- dataframes and vectors

Basic operators

```
5+5
```

```
[1] 10
```

```
2*2
```

```
[1] 4
```

```
10/2
```

```
[1] 5
```

```
3**2
```

```
[1] 9
```

```
sqrt(9)
```

```
[1] 3
```

```
3==3
```

```
[1] TRUE
```

```
3==4
```

```
[1] FALSE
```

```
10>5
```

```
[1] TRUE
```

```
10<5
```

```
[1] FALSE
```

```
10:20
```

```
 [1] 10 11 12 13 14 15 16 17 18 19 20
```

Assigning operator

Syntax: `object <- value`

Human reading: the assigning operator `<-` assigns value of the result from the operation on the right to the object on the left

- it's a kind of a "save as" function

```
new_object <- 5+5
```

```
new_object
```

```
[1] 10
```

```
my_number <- 20
my_number
```

```
[1] 20
```

```
my_number+5
```

```
[1] 25
```

```
my_other_number <- 200
my_number + my_other_number
```

```
[1] 220
```

```
my_number == 20
```

```
[1] TRUE
```

```
my_number < my_other_number
```

```
[1] TRUE
```

Creating vectors

```
my_vector <- c(1, 2, 3, 4, 5)
my_vector
```

```
[1] 1 2 3 4 5
```

```
my_vector + 10
```

```
[1] 11 12 13 14 15
```

```
my_other_vector <- c(6:10)
my_other_vector
```

```
[1]  6  7  8  9 10
```

```
my_other_vector + my_vector
```

```
[1]  7  9 11 13 15
```

```
my_other_vector[2]
```

```
[1] 7
```

Adding comments

- any code in the line after the symbol **#** will not run
- this is usefull for adding comments to your code or for "switching off" parts of the code that you don't want to run at the moment (e.g. drafts, unfinished chunks,...)

```
# this is an important comment
# the code bellow will not run:
# 10 / 2
```

```
# the code bellow will run, because it's placed before the # sign and not after
10 / 2 # this code divides 10 by 2
```

```
[1] 5
```

Exercise

Task:

1.) create one vector which contains 10 numbers from 51 to 60

2.) create another vector which contains 10 numbers from 101 to 110

3.) save the first vector as "vect_1" and second as "vect_2"

4.) subtract (*odečti*) vect_1 from vect_2 and save the results as "vect_sub"

Solution

```
vect_1 <- c(51:60)
vect_2 <- c(101:110)

vect_sub <- vect_2 - vect_1
vect_sub
```

```
 [1] 50 50 50 50 50 50 50 50 50 50
```

Basic functions and their syntax

- functions always go with parentheses `()` (*závorky*)
- functions are doing stuff
- syntax:

```
function_name(argument1 = value1, argument2 = value2, ...)
```

Examples of simple functions:

```
mean()
```

```
mean(5:10) # mean = aritmetický průměr
```

```
[1] 7.5
```

```r
a <- mean(1:10)
a
```

```
[1] 5.5
```

summary()

```r
summary(1:10)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1.00    3.25    5.50    5.50    7.75   10.00 
```

seq()

```r
my_sequence <- seq(from = 1000,  to = 2000, by = 10)
my_sequence
```

```
  [1] 1000 1010 1020 1030 1040 1050 1060 1070 1080 1090 1100 1110 1120 1130 1140
 [16] 1150 1160 1170 1180 1190 1200 1210 1220 1230 1240 1250 1260 1270 1280 1290
 [31] 1300 1310 1320 1330 1340 1350 1360 1370 1380 1390 1400 1410 1420 1430 1440
 [46] 1450 1460 1470 1480 1490 1500 1510 1520 1530 1540 1550 1560 1570 1580 1590
 [61] 1600 1610 1620 1630 1640 1650 1660 1670 1680 1690 1700 1710 1720 1730 1740
 [76] 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890
 [91] 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000
```

lenght()

```r
length(my_sequence)
```

```
[1] 101
```

sum()

```r
sum(my_sequence)
```

```
[1] 151500
```

range()

```
range(my_sequence)
```

```
[1] 1000 2000
```

Important - how R behaves

Case sensitivity

- R is case sensitive

```
new_object <- 5
NEW_OBJECT <- 10
```

```
NEW_OBJECT == new_object
```

```
[1] FALSE
```

Different ways of how to write the code

- there are different ways how to write your code, which are all correct. All depends on your taste. Just be sure all your brackets are closed and you didn't forget the comma - ,
- have a look at these three ways, all have the same results:

```
my_sequence <- seq(from = 10,  to = 20, by = 2)
my_sequence
```

```
[1] 10 12 14 16 18 20
```

```
my_sequence <- seq(
  from = 10,
  to = 20,
  by = 2
  )

my_sequence
```

```
[1] 10 12 14 16 18 20
```

In some functions, you don´t need to specify the parameters, but we recommend you to do so, at least at the beginning.

```
my_sequence <- seq(10, 20, 2)
my_sequence
```

```
[1] 10 12 14 16 18 20
```

How R behaves 2

Be carefull with the assigning operator

This chunk of code will show the result, but will not save it (assign it)

```
sum(1:10)
```

```
[1] 55
```

This chunk, on the other hand, will save your result, but will not show it.

```
my_result <- sum(1:10)
```

To show the result, you need to run the object in which you assigned it:

```
my_result
```

```
[1] 55
```

This will overwrite your original result:

```
my_result <- 100
my_result
```

```
[1] 100
```

You should therfore use unique object names for each operations:

```r
my_result <- sum(1:10)

my_new_result <- 100
```

```r
my_result
```

```
[1] 55
```

```r
my_new_result
```

```
[1] 100
```

Objects and values

- there are different types of objects and values in R, each type allows you to do different operations

Objects

- for now, it is enough to introduce the concepts of **vectors** and **dataframes**
  - **vector**
    - a list of items of the same type that are always shown (and manipulated) in the same order
  - **dataframe**
    - a table
    - has rows and columns
    - rectangular, ie. it has identical number of rows in each column.

Values

- similarly, there are many types of values - characters, numbers, factors.
- for now, all you need to know now is that if you want to do mathematic operations, you always have to check whether your numbers are really a numbers and not something else, such as characters
- function `str()` will quickly tell you what kind of object with what kind of values you have

Vectors

- use `c()` for creating simple vectors

```
cisla <- c(1, 2, 3, 4, 5)
cisla
```

```
[1] 1 2 3 4 5
```

```
stovky <- c(101:105)
stovky
```

```
[1] 101 102 103 104 105
```

- characters must be in quotation marks (*uvozovky*)

```
artefacts <- c("pottery", "dagger", "fibula", "spondylus", "dartpoint")
artefacts
```

```
[1] "pottery"   "dagger"    "fibula"    "spondylus" "dartpoint"
```

Note the difference between numbers and characters:

```
str(cisla)
```

```
 num [1:5] 1 2 3 4 5
```

- **num** = numbers

```
str(artefacts)
```

```
 chr [1:5] "pottery" "dagger" "fibula" "spondylus" "dartpoint"
```

- **chr** = characters (*znaky*)

Note: If the vector combines numbers and words, the numbers will automatically be saved as characters. It will then not be possible to perform mathematical operations on them.

```
divny_vector <- c("pottery", 1, 5, 12, 110)
str(divny_vector)
```

```
 chr [1:5] "pottery" "1" "5" "12" "110"
```

Dataframes

- you can create dataframes by binding (*vázání*) the of the same length (!!!) together
- the columns can be of different data types
- function `cbind()` binds vectors into columns (*sloupce*) and then function `as.data.frame()` change them into **dataframe**

```
df <- as.data.frame(cbind(cisla, stovky, artefacts))
df
```

```
  cisla stovky artefacts
1     1    101   pottery
2     2    102    dagger
3     3    103    fibula
4     4    104 spondylus
5     5    105 dartpoint
```

- note that the function `cbind()` was nested into function `as.data.frame()`
- nesting functions inside another demands less lines of code, but perhaps makes your code more complicated
- this is another way how to write the code with same result:

```
x <- cbind(cisla, stovky, artefacts)
df_2 <- as.data.frame(x)
```

```
df_2
```

```
  cisla stovky artefacts
1     1    101   pottery
2     2    102    dagger
3     3    103    fibula
4     4    104 spondylus
5     5    105 dartpoint
```

Dataframe - structure and types of values

Get the basic information about the dataframe with `str()`

```
str(df)
```

```
'data.frame':   5 obs. of  3 variables:
 $ cisla    : chr  "1" "2" "3" "4" ...
 $ stovky   : chr  "101" "102" "103" "104" ...
 $ artefacts: chr  "pottery" "dagger" "fibula" "spondylus" ...
```

We can see that columns *cisla* and *stovky* are not numbers, but characters. To be able for us to do mathematic operations, we need to change the values into numbers by function `as.numeric()`

Note we are use `$` to define column we need to change. We will talk about the `$` more in the next slide.

```
df$cisla <- as.numeric(df$cisla)
```

```
df$stovky <- as.numeric(df$stovky)
```

```
str(df)
```

```
'data.frame':   5 obs. of  3 variables:
 $ cisla    : num  1 2 3 4 5
 $ stovky   : num  101 102 103 104 105
 $ artefacts: chr  "pottery" "dagger" "fibula" "spondylus" ...
```

Subseting data

Square brackets `[,]`

- usefull for chosing data based on their position in dataframe or vector (number of row or column)
- impress your friends with new new English words: column = *sloupec*, row = *řádek*, square brackets = *hranaté závorky*, subseting = *vytváření podskupin, podmnožin*

Syntax: `name_of_your_dataframe[row_number, column_number]`

- subseting first row:

```
df[1,]
```

```
  cisla stovky artefacts
1     1    101   pottery
```

- subseting first column:

```
df[,1]
```

```
[1] 1 2 3 4 5
```

- applying functions on the subsets:

```
sum(df[,2])
```

```
[1] 515
```

You can even "save" subset with the help of assigning marker `<-`

```
druhy_riadok <- df[2,]
druhy_riadok
```

```
  cisla stovky artefacts
2     2    102    dagger
```

Subseting data

Subsetting with `$`

- you can use names of columns, so you don't need to remember their positions

Syntax: `name_of_your_dataframe$name_of_the_column`

```
df$artefacts
```

```
[1] "pottery"   "dagger"    "fibula"    "spondylus" "dartpoint"
```

```r
mean(df$stovky)
```

```
[1] 103
```

Now some "fun" with a fake burial ground

Copy and paste this huge piece of code, or open the script "fake_graves.R"

```r
grave_number <- 800:819

dating <- c(
  "ne.lin","ne.lin","en.zvo","en.zvo","en.snu","br.une","br.une","br.une","la.a","rstred",
  "ne.lin","br.une","en.zvo","en.snu","la.a","br.une","rstred","ne.lin","en.zvo","br.une"
)

sex <- c(
  "male","male","male","female","male","female","female","female","male","female",
  "male","female","male","female","male","male","female","female","female","male"
)

age <- c(
  "31-40","21-30","<11","31-40",">50","31-40",">50","41-50","31-40","<11",
  "<11","31-40","21-30",">50","41-50","31-40","21-30","31-40","<11",">50"
)


pottery <- c(
  3,4,3,2,5,4,5,3,2,1,
  1,6,4,7,3,5,2,5,1,6
)

bronze <- c(
  0,0,0,0,0,5,1,2,0,0,
  0,3,0,0,0,2,0,0,0,4
)

stone_chipped <- c(
  1,1,0,0,2,1,0,0,0,0,
  0,1,0,2,0,2,0,1,0,1
)
```

```
stone_polished <- c(
  2,1,0,0,1,0,0,0,0,0,
  0,0,0,1,0,0,0,1,0,0
)


grave_length <- c(
  210,160,180,250,300,200,225,250,150,100,
  90,230,200,260,210,240,180,220,100,270
)

grave_depth <- c(
  50, 40, 70,200,250,100, 80, 70, 40, 30,
  25,120, 90,180,100,150, 80,120, 30,200
)


df_grave <- as.data.frame(cbind(grave_number, dating, sex, age, pottery, bronze, stone_chippe


df_grave$pottery <- as.numeric(df_grave$pottery)
df_grave$bronze <- as.numeric(df_grave$bronze)
df_grave$stone_chipped <- as.numeric(df_grave$stone_chipped)
df_grave$stone_polished <- as.numeric(df_grave$stone_polished)
df_grave$grave_length <- as.numeric(df_grave$grave_length)
df_grave$grave_depth <- as.numeric(df_grave$grave_depth)
```

There are more elegant ways to prepare such a table. But for now, this is enough.

```
str(df_grave)
```

```
'data.frame':   20 obs. of  10 variables:
 $ grave_number  : chr  "800" "801" "802" "803" ...
 $ dating        : chr  "ne.lin" "ne.lin" "en.zvo" "en.zvo" ...
 $ sex           : chr  "male" "male" "male" "female" ...
 $ age           : chr  "31-40" "21-30" "<11" "31-40" ...
 $ pottery       : num  3 4 3 2 5 4 5 3 2 1 ...
 $ bronze        : num  0 0 0 0 0 5 1 2 0 0 ...
 $ stone_chipped : num  1 1 0 0 2 1 0 0 0 0 ...
 $ stone_polished: num  2 1 0 0 1 0 0 0 0 0 ...
 $ grave_length  : num  210 160 180 250 300 200 225 250 150 100 ...
 $ grave_depth   : num  50 40 70 200 250 100 80 70 40 30 ...
```

Let's play!

What's the dating of the graves?

```
df_grave$dating
```

```
 [1] "ne.lin" "ne.lin" "en.zvo" "en.zvo" "en.snu" "br.une" "br.une" "br.une"
 [9] "la.a"   "rstred" "ne.lin" "br.une" "en.zvo" "en.snu" "la.a"   "br.une"
[17] "rstred" "ne.lin" "en.zvo" "br.une"
```

This is bit messy, so let's use `unique()` to just get a list of dating categories present:

```
unique(df_grave$dating) # returns each **unique** value
```

```
[1] "ne.lin" "en.zvo" "en.snu" "br.une" "la.a"   "rstred"
```

Which dating group is the most represented?

- `table()` returns frequency of each value, or in other words, one-way contingency table (*kontingenční tabulka s jednou proměnnou*)

```
table(df_grave$dating)
```

```
br.une en.snu en.zvo   la.a ne.lin rstred
     6      2      4      2      4      2
```

Let´s play a bit 2

How do the graves dated to "ne.lin" look like?

In other words, we want to subset rows with graves dated to "ne.lin" (AKA *kultura s lineární keramikou*)

```
df_grave_lin <- df_grave[df_grave$dating=="ne.lin",]
df_grave_lin
```

```
   grave_number dating    sex   age pottery bronze stone_chipped stone_polished
1           800 ne.lin   male 31-40       3      0             1               2
2           801 ne.lin   male 21-30       4      0             1               1
11          810 ne.lin   male   <11       1      0             0               0
18          817 ne.lin female 31-40       5      0             1               1
   grave_length grave_depth
1           210          50
2           160          40
11           90          25
18          220         120
```

Don't worry — we'll soon learn a more intuitive way to filter and subset data.

How many chipped stone artefacts were found?

```
sum(df_grave$stone_chipped)
```

```
[1] 12
```

How many pottery pieces were found in each archaeological culture?

This looks complicated at the first sight, but don't panic:

```
aggregate(pottery ~ dating, data = df_grave, FUN = sum)
```

```
  dating pottery
1 br.une      29
2 en.snu      12
3 en.zvo      10
4   la.a       5
5 ne.lin      13
6 rstred       3
```

Human reading: "Take the variable **pottery** and compute its **sum** for each value of **dating** in the dataframe **df_grave**."

Let´s play a bit 2

What is the average length of a grave?

```
mean(df_grave$grave_length)
```

```
[1] 201.25
```

Is there difference between grave length in dating categories?

```
aggregate(grave_length ~ dating, data = df_grave, FUN = mean)
```

```
  dating grave_length
1 br.une     235.8333
2 en.snu     280.0000
3 en.zvo     182.5000
4   la.a     180.0000
5 ne.lin     170.0000
6 rstred     140.0000
```

Excercise

Task:

1. save your work, clean your workspace and open "fake_graves.R" script
2. run the code to create the dataframe "df_grave"

Answer the following questions:

3. which age group is the most represented?
4. Which category has the longest graves on average?
5. How many bronze artefacts were found?
6. which culture (dating group) had the most bronze tools?
7. subset all female graves and create "df_female_graves" object
8. what is the average number of pottery in female graves?

Results:

3. which age group is the most represented?

```
table(df_grave$age)
```

```
  <11   >50 21-30 31-40 41-50
    4     4     3     7     2
```

4. Which age group has the longest graves on average?

```
aggregate(grave_length ~ age, data = df_grave, FUN = mean)
```

```
    age grave_length
1   <11     117.5000
2   >50     263.7500
3 21-30     180.0000
4 31-40     214.2857
5 41-50     230.0000
```

5. How many bronze artefacts were found?

```
sum(df_grave$bronze)
```

```
[1] 17
```

6. Which culture (dating group) had the most bronze artefacts?

```
aggregate(bronze ~ dating, data = df_grave, FUN = sum)
```

```
  dating bronze
1 br.une     17
2 en.snu      0
3 en.zvo      0
4   la.a      0
5 ne.lin      0
6 rstred      0
```

7. subset all female graves and create “df_female_grave” object

```
df_female_grave <- df_grave[df_grave$sex == "female",]
```

8. what is the average number of pottery in female graves?

```
mean(df_female_grave$pottery)
```

```
[1] 3.6
```